Detecting Quasi-equal Clocks in Timed Automata

Marco Muñiz, Bernd Westphal, and Andreas Podelski

Albert-Ludwigs-Universität Freiburg, 79110 Freiburg, Germany
{muniz,westphal,podelski}@informatik.uni-freiburg.de

Abstract. A recent optimizations technique for timed model checking starts with a given specification of *quasi-equal* clocks. In principle, the zone graph can used to detect which clocks are quasi-equal; the construction of the zone graph would, however, defeat its very purpose (which is the optimization of this construction). In this paper, we present an abstraction that is effective for the goal of the optimization based on quasi-equal clocks: it is coarse enough to yield a drastic reduction of the size of the zone graph. Still, it is precise enough to identify a large class of quasi-equal clocks. The abstraction is motivated by an intuition about the way quasi-equalities can be tracked. We have implemented the corresponding reasoning method in the Jahob framework using an SMT solver. Our experiments indicate that our intuition may lead to a useful abstraction.

1 Introduction

Timed automata and timed model checking [1,12,18] have been very successfully applied for the verification of real-time systems. Still, the number of clocks in a timed automaton will always be an issue for scalability, and the optimization of timed model checking will always be a topic of research. Optimization techniques for timed model checking are often based on some notion of redundancy in the representation of the timed model and its behavior in terms of clock valuations. The detection of the corresponding redundancies is then a prerequisite for applying the optimization.

An example of a redundancy which gives the opportunity of a provably very effective optimization is the notion of *quasi-equal* clocks [11]. Two clocks x and y in a given timed automaton are quasi-equal if the invariant $x = y \vee x = 0 \vee y = 0$ holds. That is, in every step in every transition sequence, the two clocks x and y have equal value except for steps where one of them has been reset but not the other. As a consequence, the invariant $x = y$ can be violated only at single time points (i.e., during time periods of length zero). In a way, the violation of the invariant is an artefact of the model of the behavior of a timed systems by discrete sequences.

The optimization presented in [11] starts with a given specification of *quasi-equal* clocks. In principle, the zone graph can be used to detect which clocks are quasi-equal; the construction of the zone graph would, however, defeat its

V. Braberman and L. Fribourg (Eds.): FORMATS 2013, LNCS 8053, pp. 198–212, 2013.

very purpose (which is the optimization of this construction). In this paper, we present an abstraction that is coarse enough to yield a drastic reduction of the zone graph and precise enough to identify a large class of quasi-clocks.

The abstraction is motivated by an intuition about the way quasi-equalities can be tracked. The intuition is that the behavior of a timed automaton over a non-zero period of time (without resets) will neither introduce new quasi-equalities nor "destroy" a quasi-equality, and hence we can apply the most coarse abstraction there. In contract, when different values for quasi-equal clocks arise in a sequence of configurations where time does not elapse, we must track the constants for the values of the clocks as precisely as possible (i.e., apply no proper abstraction). Thus, as an intermediate step for computing abstract zones, we must apply logical reasoning in order to infer whether the zone accounts for behavior of zero (as opposed to: non-zero) periods of time.

Our abstraction methods amounts to computing an abstraction of the zone graph. We use the abstract zone graph to detect quasi-equal clocks.

We have implemented our method in the Jahob verification framework. This allows us to represent zones (and abstract zones) by linear real arithmetic formulas and to perform the required logical reasoning (on the duration of the corresponding period of time) through calls of an SMT solver.

We have used our implementation to conduct preliminary experiments. The results indicate that the abstraction is effective for the goal of the optimization based on quasi-equal clocks: it is coarse enough to yield a drastic reduction of the size of the zone graph. Still, the abstraction is precise enough to identify a large class of quasi-clocks.

Related work. In [13], a syntactical pattern for timed automata is proposed. Automata constructed under this pattern are called *sequential* timed automata. There, the so-called *master* clocks will be reset at exactly the same point of time. Thus, master clocks are quasi-equal *by construction*. Unfortunately, the class of sequential timed automata is rather restricted. Therefore, one would like to be able to use the general class of timed automata and apply a method for detecting or checking quasi-equal clocks. This motivates the present work.

In [7] an abstraction method is proposed for detecting *equal* clocks (at a particular location). Once equal clocks at a particular location have been detected, a substitution method can be used to reduce the number of clocks in that location. Thus, the method can effectively reduce the number of clocks per location and also in the whole timed automaton. In this sense, the method [7] is similar wrt. its goal to our method. The difference lies in technical details. Computing quasi-equal clocks requires to detect zero time paths. For this, our method tracks the actual valuations of clocks, which the method [7] does not. As a consequence, it will not be able to detect quasi-equal clocks when they are not equal.

Outline of the paper. In Section 2, we use an example to recall and illustrate the notion of quasi-equal clocks. We also investigate the issue of zero time behaviors and we present the abstract transition system for detecting quasi-equal clocks. In Section 3, we present the formal setting. In Section 4, we formalize our

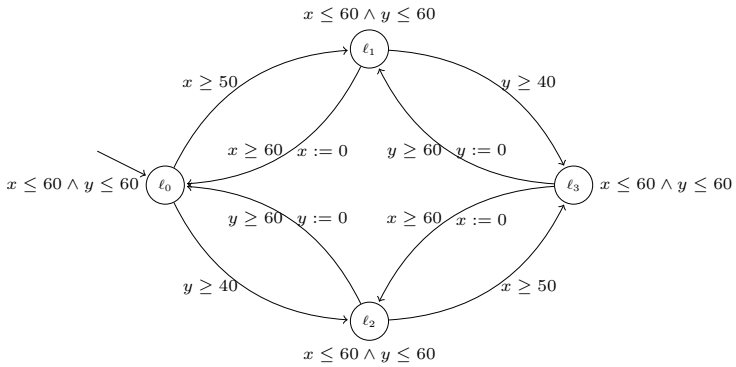

Fig. 1. Automaton with quasi-equal clocks x, y

abstraction by means of an abstract zone graph and a simulation relation. In Section 5, we present an example that illustrates effectiveness of the abstraction, i.e., the reduction of size through the abstraction. In Section 6, we present an algorithm for computing the reachable abstract zone graph (on the fly). The output of the algorithm is a relation that identifies which clocks are quasi-equal. We present the result of our experiments using an implementation of the algorithm.

2 Example

In this section, we illustrate our approach with help of the automaton in Figure 1. First, we refresh the notion of quasi-equal clocks. Next, we show the importance of the zero time behavior for detecting quasi-equal clocks. Finally, we show the corresponding abstract zone graph in which quasi-equal clocks can be efficiently detected.

Quasi-equal clocks. two clocks are quasi-equal if for all computations their values are equal or if one clock was reset then a reset must eventually occur for the other clock in zero time. Quasi-equal clocks are formally defined in Section 3. Consider the timed automata presented in Figure 1 with clocks x and y. Clearly, clocks x and y are not equal since for example in the computation $\langle \ell_0, \{\nu_0\}\rangle \rightarrow^* \langle \ell_1, x = 60 \wedge y = 0\rangle$ the configuration $\langle \ell_1, x = 60 \wedge y = 0\rangle$ yields different values for clocks x and y. However, note that the invariant $I(\ell_1) = x \leq 60 \wedge y \leq 60$ will prevent time to elapse at this configuration and thus the only successor of this configuration is $\langle \ell_0, x = 0 \wedge y = 0 \wedge x \leq 60 \wedge y \leq 60\rangle$ in which the values for clocks x and y are equal. Indeed, it is the case that for all computations from the automaton in Figure 1, the values for clocks x and y are either equal or the value of one clock is zero and a reset in zero time for the other clock occurs. Therefore, clocks x and y are quasi-equal. Clearly, the behavior of timed automata in Figure 1, can be simulated by using one clock, which yields an important speed up in the verification time.

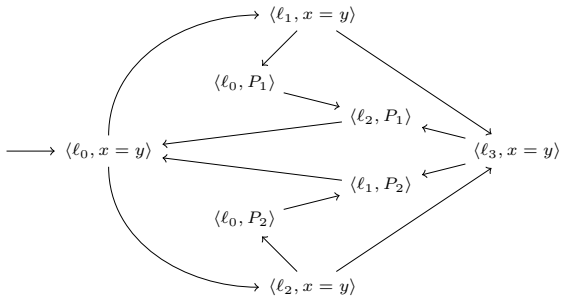

Fig. 2. Abstract zone graph corresponding to TA in Figure 1

Zero time behavior. As shown above, the zero time behavior of a timed automaton may cause quasi-equal clocks to arise. Therefore, zero time behavior is important for detecting quasi-equal clocks.

In the computation given above, first a reset for clock x occurs, then in the next transition a reset for clock y occurs. In this case, the length of the zero time computation is just one transition. However, this does not need to be the case. Consider the computation $\langle \ell_0, \{\nu_0\} \rangle \rightarrow^* \langle \ell_0, x = 0 \wedge y \leq 60 \wedge y \geq 60 \rangle$, the invariant $I(\ell_0) = x \leq 60 \wedge y \leq 60$ will not let time elapse and the only successor is $\langle \ell_2, x = 0 \wedge y \leq 60 \wedge y \geq 60 \rangle$. The invariant $I(\ell_2) = x \leq 60 \wedge y \leq 60$ will not let time elapse and the only successor is $\langle \ell_0, x = 0 \wedge y = 0 \wedge x \leq 60 \wedge y \leq 60 \rangle$, where time may elapse but the values for clocks x and y are equal. In general, the number of transitions that may occur in zero time might be infinite.

Abstract zone graph. Given a timed automaton, our method will construct an abstract transition system, which preserves as much as possible the zero time behavior of a timed automaton. The configurations of the abstract transition system are pairs consisting of locations and zones. The zones that we compute are of two types. Either a zone for which time is guaranteed not to elapse or a conjunction of equalities for clocks for which time may elapse.

Figure 2 shows the abstract transition system corresponding to the abstraction defined in Section 4 applied to the automaton in Figure 1, where zones P_1, P_2 are $P_1 := x \leq 60 \wedge x \geq 60 \wedge y = 0$ and $P_2 := y \leq 60 \wedge y \geq 60 \wedge x = 0$. Note, that in configurations where time may elapse the corresponding zone is $x = y$. In the transition induced by the edge $(\ell_1, x \geq 60, x := 0, \ell_0)$ from the automaton in Figure 1 and configuration $\langle \ell_1, P_1 \rangle$. The successor configuration $\langle \ell_0, P' \rangle$ with zone $P' = (P_1 \wedge x \geq 60)[\{x\} := 0] \wedge I(\ell_0)$ is not zero time. Therefore, $(P_1 \wedge x \geq 60)[\{x\} := 0] \wedge I(\ell_0)$ is relaxed to $x = y$, leading to configuration $\langle \ell_0, x = y \rangle$. Since, every zone in the set of reachable configurations implies that $x = y \vee x = 0 \vee y = 0$, our abstraction allow us to soundly conclude that x and y are quasi-equal.

3 Preliminaries

The formal basis for our work are timed automata [1]. Let $\mathbb{X}$ be a set of clocks. The set $\Phi(\mathbb{X})$ of simple clock constraints over $\mathbb{X}$ is defined by the grammar $\varphi ::= x \sim y \mid x - y \sim C \mid \varphi_1 \wedge \varphi_2$ where $x, y \in \mathbb{X}$, $C \in \mathbb{Q}_0^+$, and $\sim \in \{<, \leq, =, \geq, >\}$. Constraints of the form $x - y \sim C$ are called difference constraints. We assume the canonical satisfaction relation "$\models$" between *valuations* of the clocks $\nu : \mathbb{X} \to \mathbb{R}_0^+$ and simple clock constraints.

A timed automaton $\mathcal{A}$ is a tuple $(L, \mathbb{X}, I, E, \ell_0)$, which consists of a finite set of *locations* L, with typical element ℓ, a finite set of *clocks* $\mathbb{X}$, a mapping $I : L \to \Phi(\mathbb{X})$, that assigns to each location a *clock constraint*, and a set of *edges*. $E \subseteq L \times \Phi(\mathbb{X}) \times \mathcal{P}(\mathbb{X}) \times L$. An edge $e = (\ell, \varphi, Y, \ell') \in E$ from ℓ to ℓ' involves a *guard* $\varphi \in \Phi(\mathbb{X})$, and a *reset set* $Y \subseteq \mathbb{X}$. For easiness of presentation our definition of timed automaton does not include synchronizations or data variables. However, our idea can be extended to such "richer models" in a straight forward manner. For the rest of the paper, let us fix a timed automaton $\mathcal{A} = (L, \mathbb{X}, I, E, \ell_0)$.

A *zone* is the maximal set of clock valuations satisfying a clock constraint. Given timed automaton $\mathcal{A}$, for a clock constraint $Z \in \Phi(\mathbb{X})$, let $[Z]$ denote the maximal set of valuations satisfying Z. In the following we shall use Z to stand for $[Z]$ as a shorthand. Then, $\Phi(\mathbb{X})$ denotes the set of zones for $\mathcal{A}$. For zone Z, we define $Z^\uparrow = \{\nu + d \mid \nu \in Z, d \in \mathbb{R}_+\}$ and $Z[Y := 0] = \{\nu[Y := 0] \mid \nu \in Z\}$ where $\nu[Y := 0]$ denotes the valuation obtained from ν by resetting exactly the clocks in Y.

The symbolic semantics for timed automaton $\mathcal{A}$ is defined by the zone graph $\mathcal{S}(\mathcal{A}) = (\mathit{Conf}(\mathcal{A}), \to, c_0)$ where $\mathit{Conf}(\mathcal{A}) \subseteq L \times \Phi(\mathbb{X})$ is the set of configurations consisting of pairs of a location $\ell \in L$ and a zone $Z \in \Phi(\mathbb{X})$, $c_0 = \langle \ell_0, \{\nu_0\} \rangle$ is the initial configuration where $\nu_0(x) = 0$ for all clocks $x \in \mathbb{X}$ and $\to \subseteq \mathit{Conf}(\mathcal{A}) \times \mathit{Conf}(\mathcal{A})$ is the transition relation with delay transitions $\langle \ell, Z \rangle \to \langle \ell, Z^\uparrow \wedge I(\ell) \rangle$, and action transitions $\langle \ell, Z \rangle \to \langle \ell', (Z \wedge \varphi)[Y := 0] \wedge I(\ell') \rangle$ if there exists an edge $(\ell, \varphi, Y, \ell') \in E$.

The *quasi-equal* relation $\equiv$ for timed automaton $\mathcal{A}$ introduced in [11,13] is the relation containing all pairs of clocks for which in all computations of $\mathcal{A}$ their values are equal, except at points of time where they are reset and time is not allowed to elapse. Formally, it is defined as $\equiv \stackrel{\mathsf{def}}{=} \{(x, y) \mid x, y \in \mathbb{X}$ and $\langle \ell_0, \{\nu_0\} \rangle \to^* \langle \ell, Z \rangle \implies \forall \nu \in Z.\ \nu \models x = y \vee x = 0 \vee y = 0\}$. This notion is illustrated by the Example Section 2.

A *normalization operator* norm defined on zones is used to construct a finite representation of the transition relation $\to$. Maximal bound normalization [15,8], lower and upper bound zone based abstractions [3] and normalization using difference constraints [5] are some normalization procedures we may use to present our idea. Since our model for timed automata includes diagonal constraints we will define norm to be the normalization operator presented in [5]. We now formally define the norm operator. For timed automaton $\mathcal{A}$, let $\mathcal{G}$ be a finite set of difference constraints, and $k : \mathbb{X} \to \mathbb{Q}_0^+$ be a function mapping each clock x to the maximal constant $k(x)$ appearing in the guards or invariants in $\mathcal{A}$ containing x. For a real d let $\{d\}$ denote the fractional part of d and $\lfloor d \rfloor$ denote its

integer part. Two valuations ν, ν' are equivalent, denoted $\nu \sim \nu'$ iff (1) for all x, either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or both $\nu(x) > k(x)$ and $\nu'(x) > k(x)$,(2) for all x, if $\nu(x) \leq k(x)$ then $\{\nu(x)\} = 0$ iff $\{\nu(x)\} = 0$ and (3) for all x, y if $\nu(x) \leq k(x)$ and $\nu(y) \leq k(y)$ then $\{\nu(x)\} \leq \{\nu(y)\}$ iff $\{\nu(x)\} \leq \{\nu'(x)\} \leq \{\nu'(y)\}$ (4) for all $\varphi \in \mathcal{G}$, $\nu \in \varphi$ iff $\nu' \in \varphi$. Then $\mathsf{norm}(Z) \stackrel{\mathsf{def}}{=} \{\nu \mid \nu \sim \nu', \nu' \in Z\}$.

4 Zero Time Behavior Abstraction

In this section, we present our method for detecting quasi-equal clocks. The main observation is that if two clocks x and y are quasi-equal then for all computations if one clock, say x is reset then a reset for the other clock y must appear in some future configuration in the computation path. In particular, for all the configurations in the computation fragment between the resets of x and y time cannot elapse (i.e. all the delay successors of a configuration have a delay of zero). Another key observation is that if in a configuration time is allowed to elapse, clocks x and y must be strictly equal. Therefore, to detect quasi-equal clocks. We do not only need to consider resets of clocks but also configurations in which time is allowed to elapse and configurations in which time is not allowed to elapse. The following definition formalizes our notion of zero time configurations, i.e. configurations for which time cannot elapse.

Definition 1 (Zero time configuration). *A configuration $\langle \ell, Z \rangle$ is* zero time *if the invariant of location ℓ precludes time to elapse. Formally,*

$$\mathsf{zt}(\ell, Z) \stackrel{\mathsf{def}}{=} \forall \nu \in Z, d \in \mathbb{R}_0^+.\ \nu + d \models I(\ell) \implies d = 0.$$

Our method preserves as much as possible the information corresponding to zero time configurations and abstracts away much information from the non-zero time configurations. If a configuration $\langle \ell, Z \rangle$ is zero time, our method will preserve all the information in Z. However, if the configuration $\langle \ell, Z \rangle$ is non-zero time, our method will abstract Z by means of the relax operator rlx to a much bigger zone $\mathsf{rlx}(Z)$, which preservers the strict equalities entailed by the zone Z. The following definition formalizes the relax operator.

Definition 2 (Relax operator). *Given a zone $Z \in \Phi(\mathbb{X})$. The* relax operator rlx *applied to the zone Z over-approximates Z by a conjunction of the clock equalities it entails. Formally,*

$$\mathsf{rlx}(Z) \stackrel{\mathsf{def}}{=} \bigwedge \{x = y \mid x, y \in \mathbb{X} \text{ and } \forall \nu \in Z.\ \nu \models x = y\}.$$

As an example of the relax operator, consider the zone Z in Figure 3 left. It is the case that all the valuations in Z satisfy the constraint $x = y$. However, there are valuations in Z which satisfy $x \neq z$ and also valuations which satisfy $y \neq z$. Therefore, the result of applying the relax operator to Z yields the zone $x = y$ as shown in Figure 3 right. Note, that the zone $\mathsf{rlx}(Z)$ has no constraints on the unequal clocks. Another important property of $\mathsf{rlx}(Z)$ is that it contains only

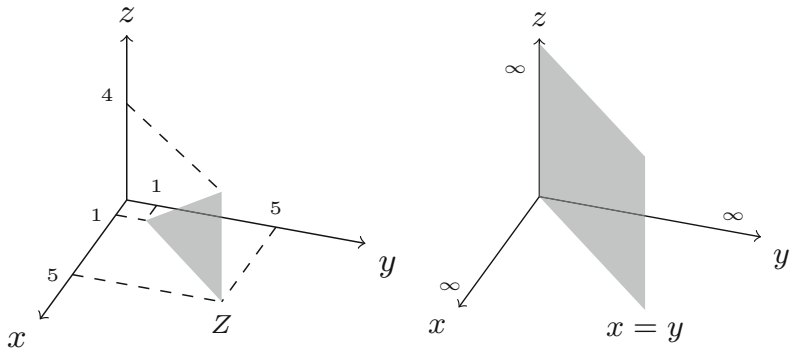

Fig. 3. The relax operator on zone $Z := x - y \leq 0 \wedge x - y \geq 0 \wedge x \geq 1 \wedge y \leq 5 \wedge y - z \leq 1$.

positive equalities and thus will remain unaffected by passage of time. That is $\mathsf{rlx}(Z) = \mathsf{rlx}(Z)^{\uparrow}$. Note, that $\mathsf{rlx}(Z)$ is much bigger than Z and it is closed with respect to delays. If a configuration is zero time our method will abstract the configuration by means of the normalization norm operator. On the contrary, if a configuration is non-zero time, our method will abstract the configuration by means of the relax rlx operator. The normalization operator norm is increasing, idempotent and yields a finite number of zones. Since our method relies on both the normalization operator norm and the relax operator rlx it is important that the relax operator exhibits the above mentioned properties.

Lemma 1. *The relax operator is increasing, idempotent and yields a finite abstraction. Formally, the following hold:*

- $Z \subseteq \mathsf{rlx}(Z)$ *for any* $Z \in \Phi(\mathbb{X})$
- $\mathsf{rlx}(\mathsf{rlx}(Z)) = \mathsf{rlx}(Z)$ *for any* $Z \in \Phi(\mathbb{X})$
- *the set* $\{\mathsf{rlx}(Z) \mid Z \in \Phi(\mathbb{X})\}$ *is finite.*

Our goal is to construct an abstract zone graph in which quasi-equalities can be soundly and efficiently computed. We now continue with the formal definition of our method.

Definition 3 (Abstract zone graph). *Timed automaton* $\mathcal{A}$ *induces the abstract zone graph* $\mathcal{S}^{\#}(\mathcal{A}) = (\mathit{Conf}(\mathcal{A}), \rightsquigarrow, c_0^{\#})$ *where:*

- $\mathit{Conf}(\mathcal{A}) \subseteq L \times \Phi(\mathbb{X})$ *is the set of configurations*
- $c_0^{\#} = (\ell_0, \bigwedge_{x,y \in \mathbb{X}} x = y)$ *is the initial configuration*
- $\rightsquigarrow \subseteq \mathit{Conf}(\mathcal{A}) \times \mathit{Conf}(\mathcal{A})$ *is the transition relation defined as:*

$$\langle \ell, P \rangle \rightsquigarrow \begin{cases} \langle \ell', \mathsf{norm}(F) \rangle & \text{if } \mathsf{zt}(\ell', F) \\ \langle \ell', \mathsf{rlx}(F) \rangle & \text{otherwise} \end{cases}$$

if there is an edge $(\ell, \varphi, Y, \ell') \in E$, *where* $F = (P \wedge \varphi \wedge I(\ell))[Y := 0] \wedge I(\ell')$.

The initial abstract configuration is a conjunction asserting all clocks to be equal. Given a configuration and an edge, the successor zone F is computed. If the destination configuration is zero time, the zone F will be normalized to $\mathsf{norm}(F)$. This is because in zero time it is important to preserve as much information as possible. This information is useful for detecting zero time paths in which quasi-equal clocks may arise. If the destination configuration is not zero time, the zone F will be relaxed to $\mathsf{rlx}(F)$. The zone $\mathsf{rlx}(F)$ consist of conjuncts asserting all strict equalities of clocks in zone F. In addition, the relax operator will remove inequalities introduced by clock resets. Note that the abstract zone graph only performs discrete transitions. This is because if a configuration is "detected" as non-zero time, the relax operator will be applied and the corresponding zone is closed with respect to delays. For easiness of presentation we use $\mathsf{norm}(F)$ to ensure the relation $\rightsquigarrow$ to be finite. We remind the reader that any normalization operator norm' such that $\mathsf{norm}'(Z) \supseteq Z$, $\mathsf{norm}'(\mathsf{norm}'(Z)) = \mathsf{norm}'(Z)$ and the set $\{\mathsf{norm}'(Z) \mid Z \in \Phi(\mathbb{X})\}$ is finite, will be suitable for our method.

For a timed automaton $\mathcal{A}$, we formalize the behavior of the corresponding abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ with respect to the zone graph $\mathcal{S}(\mathcal{A})$ via a simulation relation.

Definition 4 (Simulation relation). *Given a timed automaton $\mathcal{A}$, a simulation relation for the zone graph $\mathcal{S}(\mathcal{A}) = (Conf(\mathcal{A}), \rightarrow, c_0)$ and the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A}) = (Conf(\mathcal{A}), \rightsquigarrow, c_0^{\#})$, is a binary relation $\preccurlyeq$ on $Conf(\mathcal{A})$ such that:*

1. $c_0 \preccurlyeq c_0^{\#}$
2. *if $\langle \ell, Z \rangle \preccurlyeq \langle \ell_1, P \rangle$ then:*
   (a) $\ell = \ell_1$ *and* $Z \subseteq P$
   (b) *if $\langle \ell, Z \rangle \rightarrow \langle \ell', Z' \rangle$ with edge $(\ell, \varphi, \gamma, \ell') \in E$, then there exists $\langle \ell', P' \rangle$ such that $\langle \ell, P \rangle \rightsquigarrow \langle \ell', P' \rangle$ with edge $(\ell, \varphi, \gamma, \ell')$ and $\langle \ell', Z' \rangle \preccurlyeq \langle \ell', P' \rangle$*
   (c) *if $\langle \ell, Z \rangle \rightarrow \langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle$, then $\langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle \preccurlyeq \langle \ell, P \rangle$.*

If a simulation relation $\preccurlyeq$ exists, we say that the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ simulates the zone graph $\mathcal{S}(\mathcal{A})$.

In the rest of the paper we will consistently use Z and P to refer to zones in the zone and abstract zone graph respectively. The definition of simulation relation given above is quite specific. This allow us to better explain the relation between the zone graph and the abstract zone graph. If two configurations are in relation, the zone in the abstract zone graph is always bigger or equal than the corresponding zone in the zone graph. If there is a discrete transition in the zone graph then there is a discrete transition in the abstract zone graph as well. If there is a delay transition in the zone graph, meaning that the configuration is non-zero time, the abstract zone graph "remains" at its current configuration. In the latter case, since zones in the abstract zone graph are bigger or equal than zones in the zone graph, the corresponding abstract configuration is also non-zero time and closed with respect to delays. The following lemma guarantees the existence of a simulation relation.

Lemma 2. *For a timed automaton $\mathcal{A}$, the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ simulates the zone graph $\mathcal{S}(\mathcal{A})$.*

Our goal is to find the set of quasi-equal clocks for a given timed automaton $\mathcal{A}$. We now define the quasi-equal relation induced by the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ as the set of quasi-equalities implied by all the zones in the set of its reachable configurations.

Definition 5. *Given timed automaton $\mathcal{A}$. The abstract quasi-equal relation $\equiv^{\#}$ induced by the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$, is the set of pairs of clocks such that for all pairs, their values in the reachable configurations are equal or one clock is equal to zero. Formally,*

$$\equiv^{\#} \stackrel{\mathsf{def}}{=} \{(x,y) \mid x,y \in \mathbb{X} \text{ and } c_0^{\#} \rightsquigarrow^* \langle \ell, P\rangle \implies P \subseteq \{\nu \mid \nu \models x = y \vee x = 0 \vee y = 0\}\}.$$

Our method is sound in the sense that if two clocks are quasi-equal in the abstract zone graph, then they are quasi-equal in the concrete zone graph. Our method is not complete in the sense that if two clocks are quasi-equal in the zone graph, then they might not be quasi-equal in the abstract zone graph. As an example consider the timed automaton $\mathcal{A}' = (L, \mathbb{X}, I, E, \ell_0)$ with $L = \{\ell_0, \ell_1, \ell_2\}$, $\mathbb{X} = \{x, y\}$, $I(\ell) = \top$ for all $\ell \in L$ and $E = \{(\ell_0, x \geq 5, \{\}, \ell_1), (\ell_1, x \leq 2, \{x\}, \ell_2)\}$. Then $x \equiv y$ but $x \not\equiv^{\#} y$. This is because the edge $e = (\ell_1, x \leq 2, \{x\}, \ell_2)$ is an unfeasible edge, i.e. it does not induce a reachable transition in the zone graph. Edge e will cause clocks x, y to be unequal in the abstract zone graph. Surprisingly, we have not been able to find an example for which the abstraction is not complete given that the considered timed automaton contains only feasible edges. The following theorem states formally the soundness of the abstraction.

Theorem 1 (Zero time abstraction is sound). *Given timed automaton $\mathcal{A}$. If two clocks are quasi-equal in the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$, then they are quasi-equal in the zone graph $\mathcal{S}(\mathcal{A})$. Formally,*

$$\forall x, y \in \mathbb{X}.\ x \equiv^{\#} y \implies x \equiv y.$$

For implementation purposes, it is important that the relation $\rightsquigarrow$ is finite. Given a timed automaton $\mathcal{A}$ and by definition of the transition relation $\rightsquigarrow$. A configuration has two possible successors. Either a successor corresponding to the application of the norm operator, or a successor corresponding to the application of the rlx operator. The set of zones generated by norm is finite. Further, the set of zones generated by rlx is in $\mathcal{O}(2^{|\mathbb{X}|})$, since it contains only positive equalities for the clocks in $\mathbb{X}$. Thus, we obtain the following theorem.

Theorem 2. *The transition relation $\rightsquigarrow$ is finite.*

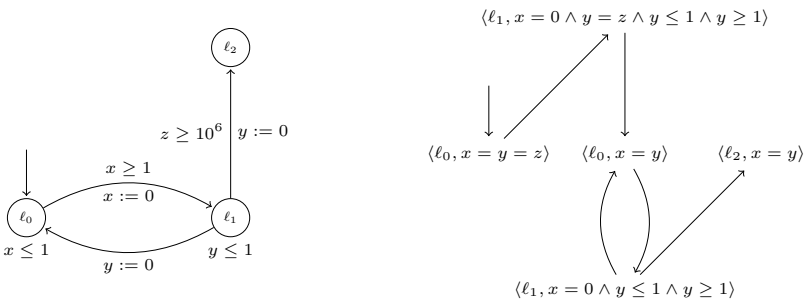

Fig. 4. Left: a timed automaton with clocks, x, y and z. Clocks x, y are quasi-equal. Right: the corresponding abstract zone graph.

5 Size of the Abstract Zone Graph

In this section we characterize the size of the abstract zone graph based on the number of reachable configurations. We show that when an automaton $\mathcal{A}$ contains only feasible edges i.e. for each edge e in the timed automaton there exists a computation path in which the edge e induces a transition, then the size of the zone graph $\mathcal{S}(\mathcal{A})$ is an upper bound for the size of the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$.

Definition 6 (Size). *For timed automaton $\mathcal{A}$ we defined the size of the zone graph $\mathcal{S}(\mathcal{A}) = (Conf(\mathcal{A}), \rightarrow, c_0)$ and the size of the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A}) = (Conf(\mathcal{A}), \rightsquigarrow, c_0^{\#})$ to be the number of reachable configurations. Formally, $|\mathcal{S}(\mathcal{A})| = |\{c \mid c_0 \rightarrow^* c\}|$ and $|\mathcal{S}^{\#}(\mathcal{A})| = |\{c \mid c_0^{\#} \rightsquigarrow^* c\}|$ respectively.*

The following theorem shows that when in the input automaton all edges induce a reachable configuration in the zone graph $\mathcal{S}(\mathcal{A})$. Then the number of configurations from the corresponding abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ is smaller or equal than the number of configurations in the zone graph $\mathcal{S}(\mathcal{A})$. The theorem follows from the following facts. First, if the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ performs an action transition with edge e then the zone graph $\mathcal{S}(\mathcal{A})$ also performs a transition with edge e. Second, by Lemma 2 it is the case that $\mathcal{S}^{\#}(\mathcal{A})$ simulates $\mathcal{S}(\mathcal{A})$ and all zones in the configurations of $\mathcal{S}^{\#}(\mathcal{A})$ are bigger than the corresponding ones in $\mathcal{S}(\mathcal{A})$.

Theorem 3 ($|\mathcal{S}^{\#}(\mathcal{A})| \leq |\mathcal{S}(\mathcal{A})|$). *Given a timed automaton $\mathcal{A}$, the zone graph $\mathcal{S}(\mathcal{A}) = (Conf(\mathcal{A}), \rightarrow, c_0)$ and the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A}) = (Conf(\mathcal{A}), \rightsquigarrow, c_0^{\#})$. If $\mathcal{A}$ is such that for all $e = (\ell, \varphi, Y, \ell') \in E$ there is a transition $\langle \ell, Z \rangle \rightarrow \langle \ell', Z' \rangle$ with edge e and $c_0 \rightarrow^* \langle \ell, Z \rangle$ then $|\mathcal{S}^{\#}(\mathcal{A})| \leq |\mathcal{S}(\mathcal{A})|$.*

The feasibility of edges is not a strong condition, since in practice unfeasible edges will not occur intentionally, except in the cases when the modeler makes a mistake. Theorem 3 gives an upper bound on the size of the abstract zone graph. In practice for a timed automaton the difference on the size of the abstract and

Algorithm 1. High level algorithm for detecting quasi-equal clocks

Input: timed automaton $\mathcal{A} = (L, \mathbb{X}, I, E, \ell_0)$
Output: a binary relation $QE \subseteq \mathbb{X} \times \mathbb{X}$ containing quasi-equal clocks in $\mathcal{A}$

```
W := {c_0^#}, V := ∅
QE := {(x, y) | x, y ∈ 𝕏 and x is different than y}
while W ≠ ∅ and QE ≠ ∅ do
    take ⟨ℓ, P⟩ from W
    for all (x, y) ∈ QE do
        if P ⋢ {ν | ν ⊨ x = y ∨ x = 0 ∨ y = 0} then
            QE := QE \ {(x, y)}
        end if
    end for
    if P ⋢ P' for all ⟨ℓ, P'⟩ ∈ V then
        add ⟨ℓ, P⟩ to V
        for all ⟨ℓ', P'⟩ with ⟨ℓ, P⟩ ⇝ ⟨ℓ', P'⟩ do
            add ⟨ℓ', P'⟩ to W
        end for
    end if
end while
return QE
```

the size of the concrete system can be exponential. To illustrate this, consider the automaton $\mathcal{A}$ in Figure 4 left. We observe that the zero time behavior occurs at points of time where $x = 1 \wedge y = 1 \wedge z = n$ with $n \in \{1, 2, \ldots, 10^6\}$. The normalized zone graph using maximal constant over approximation will contain at least 10^6 configurations. In Figure 4 right the complete full abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ is illustrated. At point of time $x = 1 \wedge y = 1 \wedge z = 1$ our method detects a zero time configuration and computes the exact successor zone $Z_1 := x = 0 \wedge y = z \wedge y \leq 1 \wedge y \geq 1$. Note that $\mathsf{zt}(\ell_1, Z_1)$ is valid, then there is a transition with edge $(\ell_1, \top, \{y\}, \ell_0)$. Our method computes $F := x = 0 \wedge y = 0 \wedge z = 1$ and the formula $\mathsf{zt}(\ell_0, F)$ is not valid, thus F is relax to $\mathsf{rlx}(F) := x = y$ leading to configuration $\langle \ell_0, x = y \rangle$. Since F does not imply a quasi-equality for clock z, the clock z is abstracted away. The resulting abstraction has only 5 configurations.

6 Algorithm and Experiments

In this section, first we present a high level algorithm for performing a reachability analysis on the abstract zone graph induced by a given timed automaton. Next, we give some details on our implementation and compare the results obtained by our implementation to the ones obtained by using a model checker.

6.1 Algorithm for Detecting Quasi-equal Clocks

Algorithm 1 lists a high level algorithm for finding quasi-equal clocks in a given timed automaton. The idea of Algorithm 1 is to traverse the reachable state

space of $\mathcal{S}^{\#}(\mathcal{A})$ while maintaining a relation QE containing quasi-equal clocks. The reachable state space is computed on the fly.

We continue by describing Algorithm 1 in detail. At line 2 the set QE is initialized to have all non reflexive quasi-equalities. At line 3 the while condition ensures that the algorithm will terminate either when we have visited the whole reachable space of $\mathcal{S}^{\#}(\mathcal{A})$ or when there are no quasi-equal clocks in QE. At lines 4 to 8 the algorithm picks a configuration $\langle \ell, P \rangle$ to be explored and checks that all the quasi-equalities (x, y) in QE are implied by P. If this is not the case, then it will remove (x, y) from QE. Note that in the algorithm the size of QE only decreases. Finally, at lines 10 to 14 the algorithm computes the successor of $\langle \ell, P \rangle$ using the transition relation $\rightsquigarrow$.

Note, that all the operations needed in Algorithm 1 can be implemented using difference bound matrices [4,10] and thus our approach can be implemented in tools like Kronos [18] or Uppaal [12]. The check in line 6 for two clocks can be implemented by checking independently whether P satisfies any disjunct in $x = y \vee x = 0 \vee y = 0$. For computing the successor in line 12 by definition of $\rightsquigarrow$ it is necessary to compute $\mathsf{zt}(\ell', F)$ where F is a convex zone. This can be computed by checking the inclusion $F \wedge I(\ell') \supseteq F^{\uparrow} \wedge I(\ell')$.

If the set QE is never empty, Algorithm 1 computes the reachable space of $\mathcal{S}^{\#}(\mathcal{A})$ and if a pair of clocks (x, y) are in QE by Theorem 1 it follows that they are quasi-equal in $\mathcal{A}$.

Theorem 4 (Partial correctness). *For any timed automaton $\mathcal{A}$, if two clocks are in relation QE from Algorithm 1, then they are quasi-equal in the corresponding zone graph. Formally,*

$$\forall x, y \in \mathbb{X}.(x, y) \in QE \implies x \equiv y.$$

Algorithm 1 will terminate whenever the relation QE is empty or whenever the wait list W is empty. By Theorem 2, the relation $\rightsquigarrow$ is finite, meaning that the list W will be eventually empty. Thus we obtain the following theorem.

Theorem 5 (Termination). *Algorithm 1 terminates for all inputs.*

6.2 Experiments

As a proof of concept we have implemented our approach in our prototype tool *Saset*. Saset is implemented in the Jahob [19,16] verification system. Our implementation represents zones as linear real arithmetic formulae and uses logical implications to ensure a finite number of representatives. As Saset constructs the abstract zone graph a number of constraints will arise, Saset will use an SMT solver to solve them. In our experiments we used the solver Z3 [9].

In general our results show that the size of the transition system computed using our abstraction is very small in comparison to the one computed by Uppaal. Therefore, the verification times for Saset are fast in spite of the number of SMT calls which are time costly.

Table 1. Results for detecting quasi-equal clocks using tools Saset and Uppaal. Saset returns the set of quasi-equal clocks whereas Uppaal performs a single query asserting clocks to be quasi-equal. Note, that for detecting quasi-equal clocks multiple queries are needed. The zero time behavior for automata in classA remains constant, in classB grows linearly and in classC grows exponentially.

Automaton	clocks	qe-clocks	max k	Saset			Uppaal		
				SMT-calls	states	t (s)	Q	states	t (s)
classA1	3	2	10^4	26	5	0.3	φ_1	20k	7.3
classA2	3	2	10^5	26	5	0.3	φ_1	200k	1200
classA3	3	2	10^6	26	5	0.3	φ_1	t.o.	t.o.
classB2	4	2	10^4	72	8	0.8	φ_1	20k	7.4
classB3	5	3	10^4	105	10	1.3	φ_2	30k	12.5
classB4	6	4	10^4	144	12	2.2	φ_3	40k	21.0
classB5	7	5	10^4	193	14	3.5	φ_4	50k	34.8
classB6	8	6	10^4	248	16	5.16	φ_5	60k	45.2
classC2	3	2	5000	63	7	1.2	φ_1	5k	5.2
classC3	4	3	5000	237	14	8.3	φ_2	35k	31.8
classC4	5	4	5000	809	26	44.05	φ_3	75k	202.3
classC5	6	5	5000	2389	47	195.3	φ_3	150k	1007.3
classC6	7	6	5000	8515	85	844	φ_4	t.o.	t.o.

In Table 1, we present a number of results obtained by using our tool and the model checker Uppaal [12]. Our intention is not to outperform Uppaal in terms of time but to show that the abstraction method that we propose for detecting quasi-equalities is a good abstraction. Thus, we encourage the reader to focus on the number of states generated by the tools for each automaton. Note, that for Uppaal to detect n clocks to be quasi-equal it would need to perform 2^n queries whereas our tool compute them directly. In Table 1, max k is the number of the maximal constant appearing in the corresponding timed automaton and the queries φ are TCTL formulae asserting a number of clocks to be quasi-equal, e.g. $\varphi_1 := \mathsf{AG}\ x_0 = x_1 \vee x_0 = 0 \vee x_1 = 0$, $\varphi_2 := \mathsf{AG}\ (x_0 = x_1 \vee x_0 = 0 \vee x_1 = 0) \wedge (x_0 = x_2 \vee x_0 = 0 \vee x_2 = 0)$. The experiments were executed in a AMD Phenom II X6 3.2Ghz Processor with 8GB RAM running Linux 3.2.

We use three classes of timed automata which are relevant for our abstraction. For classA the zero time behavior is constant and the non zero time behavior grows. For classB the zero time behavior grows linear and the non-zero time behavior remains constant. For classC, the zero time behavior grows exponentially.

The automata in classA correspond to the automaton in Figure 4. We only change the maximal constant appearing in the automaton and observe that the size of the abstraction remains the same. For the automata in classB the number of quasi-equal clocks is increased but there is an order on the reset of the quasi-equal clocks. We observe a linear increase in the number of states for both tools. For the automata in classC the number of quasi-equal clocks is increased

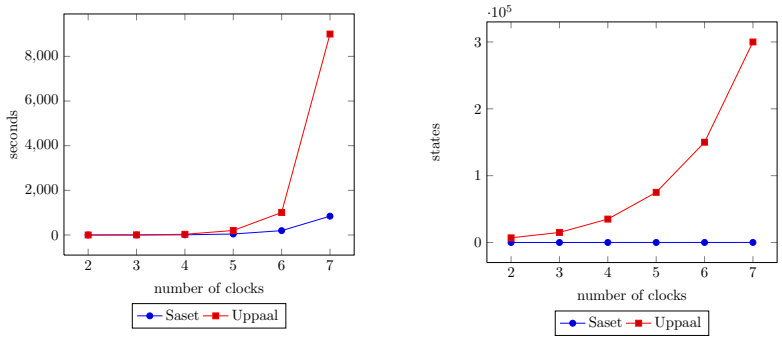

Fig. 5. Verification times and number of states explored for automata in classC

but there is no order in the reset of the clocks. We observe an exponential increase in the number of states for both tools. In Table 1, the construction of the abstraction for automaton classC6 required 8515 SMT calls, according to our tool these SMT calls took 695.2 seconds, which is 82% of the time cost. Thus, a more efficient implementation is desirable. Since our algorithm can be implemented using difference bound matrices a much efficient implementation is possible. Figure 5 illustrates the results for automata in classC.

Once the quasi-equal clocks in a timed automaton have been detected. A reduction on the number of clocks might take place, leading to a major speed up. As an example we have reduced all the quasi-equal clocks for the automaton classC6 by replacing them with a representative clock. The resulting zone graph computed by Uppaal consists of 5003 states and invariant properties can be verified in few seconds, which is an exponential gain.

7 Conclusion

In this paper, we have presented an abstraction that is effective for the goal of an already established optimization technique which is based on quasi-equal clocks. The abstraction is motivated by an intuition about the way quasi-equalities can be tracked. We have implemented the corresponding reasoning method in the Jahob framework using an SMT solver. Our experiments indicate that our intuition may lead to a useful abstraction. I.e., the is coarse enough to yield a drastic reduction of the size of the zone graph. Still, it is precise enough to identify a large class of quasi-clocks.

The goal of our prototypical implementation is to be able to evaluate the effectiveness of our abstraction for the goal of the optimization. An orthogonal issue is the optimization of the execution time of the abstraction method itself. In our experiments, the execution time is acceptable. Possibly the execution time can be improved by exchanging the general-purpose SMT solver with a specialized machinery, i.e., difference bound matrices [4,10]. We leave this aspect to future work.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Behrmann, G., Bouyer, P., Larsen, K.G., Pelánek, R.: Lower and upper bounds in zone based abstractions of timed automata. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 312–326. Springer, Heidelberg (2004)
3. Behrmann, G., Bouyer, P., Larsen, K.G., Pelánek, R.: Lower and upper bounds in zone-based abstractions of timed automata. International Journal on Software Tools for Technology Transfer 8(3), 204–215 (2006)
4. Bellman, R., Kalaba, R.E.: Dynamic programming and modern control theory. Academic Press, New York (1965)
5. Bengtsson, J., Yi, W.: On clock difference constraints and termination in reachability analysis of timed automata. In: Dong, J.S., Woodcock, J. (eds.) ICFEM 2003. LNCS, vol. 2885, pp. 491–503. Springer, Heidelberg (2003)
6. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)
7. Daws, C., Yovine, S.: Reducing the number of clock variables of timed automata. In: Proc. RTSS 1996, pp. 73–81. IEEE Computer Society Press (1996)
8. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 313–329. Springer, Heidelberg (1998)
9. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
10. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
11. Herrera, C., Westphal, B., Feo-Arenis, S., Muñiz, M., Podelski, A.: Reducing quasi-equal clocks in networks of timed automata. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 155–170. Springer, Heidelberg (2012)
12. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer (STTT) 1(1), 134–152 (1997)
13. Muñiz, M., Westphal, B., Podelski, A.: Timed automata with disjoint activity. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 188–203. Springer, Heidelberg (2012)
14. Olderog, E.R., Dierks, H.: Real-Time Systems - Formal Specification and Automatic Verification. Cambridge University Press (2008)
15. Pettersson, P.: Modelling and verification of real-time systems using timed automata: theory and practice. Ph.D. thesis, Citeseer (1999)
16. Podelski, A., Wies, T.: Counterexample-guided focus. In: Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, pp. 249–260. ACM, New York (2010), http://doi.acm.org/10.1145/1706299.1706330
17. Wies, T., Muñiz, M., Kuncak, V.: An efficient decision procedure for imperative tree data structures. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 476–491. Springer, Heidelberg (2011)
18. Yovine, S.: Kronos: A verification tool for real-time systems. International Journal on Software Tools for Technology Transfer (STTT) 1(1), 123–133 (1997)
19. Zee, K., Kuncak, V., Rinard, M.: Full functional verification of linked data structures. In: Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2008, pp. 349–361. ACM, New York (2008), http://doi.acm.org/10.1145/1375581.1375624